```
import torch
import torchvision as tv

B = 100
transf = tv.transforms.Compose(
            [ tv.transforms.ToTensor(),
              tv.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
            ])                      # ( value - mean ) / std

trn_data = tv.datasets.CIFAR10( root='./data', train=True,
                            download=True, transform=transf)
tst_data = tv.datasets.CIFAR10( root='./data', train=False,
                            download=True, transform=transf)

trn_load = torch.utils.data.DataLoader( trn_data, batch_size=B,
                                    shuffle=True, num_workers=2)
tst_load = torch.utils.data.DataLoader( tst_data, batch_size=B,
                                    shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird',  'cat',  'deer',
           'dog', ' frog', 'horse', 'ship', 'truck')
C = len(classes)

# idata = iter( tst_load)
# image, label = next( idata)

class ConvNet( torch.nn.Module):
    def __init__( _, num_classes):          # O = | (I + 2p - k)/s + 1 |
        super().__init__()                  # Tambien pueden ser tuplas.
        _.conv1 = torch.nn.Conv2d(  3, 16, kernel_size=5,
                                              stride=2, padding=2)
        _.conv2 = torch.nn.Conv2d( 16, 32, kernel_size=3,
                                              stride=1, padding=1)
        _.mpool = torch.nn.MaxPool2d( kernel_size=2, stride=2)
        _.hlin1 = torch.nn.Linear( 32*8*8, 512)
        _.hlin2 = torch.nn.Linear( 512, num_classes)

    def forward( _, x):                         # [B,  3, 32, 32]
        h1 = _.conv1( x).relu()                 # [B, 16, 16, 16]
        h2 = _.mpool( _.conv2( h1)).relu()      # [B, 32,  8,  8]
        h2 = h2.view( -1, 32*8*8)               # [B, 32 * 8 * 8]
        h3 = _.hlin1( h2).tanh()                # [B, 512]
        y  = _.hlin2( h3)                       # [B, 10]
        return y
```

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
T = 5

model = ConvNet( C).to( device)
optim = torch.optim.Adam( model.parameters())
costf = torch.nn.CrossEntropyLoss()

model.train()
for t in range(T):
    for i, (images, labels) in enumerate(trn_load):
        images = images.to( device)
        labels = labels.to( device)
        output = model( images)
        error = costf( output, labels)
        optim.zero_grad()
        error.backward()
        optim.step()
        if (i+1)%100 == 0:
            print ( "Epoch [{}/{}], Batch [{}], Loss: {:.4f}"
                    .format( t+1, T, i+1, error.item()))

model.eval()
c, t = 0, 0
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in tst_load:
        images = images.to( device)
        labels = labels.to( device)
        output = model( images)
        _, predicted = torch.max( output.data, 1)
        c += (predicted == labels).sum().item()
        t += labels.size(0)

print( "Accuracy: {} %".format(100*c/t))
torch.save( model, 'convnet.model')

# _.enc = nn.Sequential( nn.Conv2d(3, 6, kernel_size=5), nn.ReLU(True),
#                        nn.Conv2d(6,16, kernel_size=5), nn.ReLU(True) )
# _.dec = nn.Sequential( nn.ConvTranspose2d(16,6, 5), nn.ReLU(True),
#                        nn.ConvTranspose2d( 6,3, 5), nn.Sigmoid() )
```